Learning DAX: How to Extract Substrings in Power BI

Authored by

Mohammed looti

November 12, 2025

Introduction to Text Manipulation and Substrings in Power BI

In the realm of modern business intelligence, data often arrives in an unstructured or semi-structured format. A fundamental skill for any data analyst working in Power BI is the ability to efficiently parse and segment text strings. This process, known as substring extraction, is essential for cleaning data, creating meaningful features, and isolating identifiers from complex fields like URLs, product codes, or email addresses. Without the capability to precisely target and retrieve these specific textual components, analysts would be severely limited in their ability to perform detailed reporting and analysis.

In the Power BI ecosystem, text manipulation is predominantly handled through DAX (Data Analysis Expressions). DAX is the formula language used throughout Power BI, Analysis Services, and Power Pivot in Excel. While Power Query (M language) is often used for initial data transformation, DAX calculated columns and measures provide powerful, row-context-aware methods for extracting and manipulating data directly within the data model. Mastering these DAX text functions is crucial for any advanced data modeler seeking to derive maximum value from raw string data.

This guide focuses specifically on the core DAX functions designed for substring extraction. We will explore both simple positional methods, which rely on fixed character counts, and advanced dynamic techniques, which utilize searching mechanisms to identify segments based on specific delimiters. By the end of this tutorial, you will possess five highly effective formulas that cover nearly all common substring requirements, enabling you to transform messy, concatenated text into clean, usable data fields ready for insightful analysis.

The Foundational DAX Positional Functions

The most straightforward approach to substring extraction relies on positional indexing--specifying the exact starting point and the number of characters to retrieve. These functions are highly analogous to the text functions found in traditional spreadsheet software, ensuring a smooth learning curve for users familiar with Excel. The three foundational functions are LEFT, MID, and RIGHT.

When implementing these formulas in Power BI, we typically create a new calculated column within our data model. For demonstration purposes throughout this article, we will assume all operations are performed on a table named **my_data**, targeting a column named **Email**. This column contains full email addresses, providing a perfect real-world scenario for testing various extraction methods.

Formula 1: Extract Substring from Start of String (Using LEFT)

first_three = LEFT('my_data', 3)

The LEFT function is the simplest of the positional functions, designed to retrieve characters starting from the very beginning (leftmost side) of the text string. It accepts two primary arguments: the string reference (the column name) and the number of characters you wish to extract. In the example above, the formula pulls the first three characters from every entry in the **Email** column. This technique is often employed when attempting to standardize short codes or extract initials.

Formula 2: Extract Substring from Middle of String (Using MID)

mid = MID('my_data', 2, 4)

The MID function offers the greatest precision for positional extraction, as it allows the user to specify both the starting point and the length of the extracted segment. This function requires three arguments: the target string, the starting position (where 1 is the first character), and the number of characters to return. In this illustration, the formula instructs DAX to begin counting from the second position and extract a total of four characters, making it highly versatile for parsing fixed-width data fields embedded within a larger string.

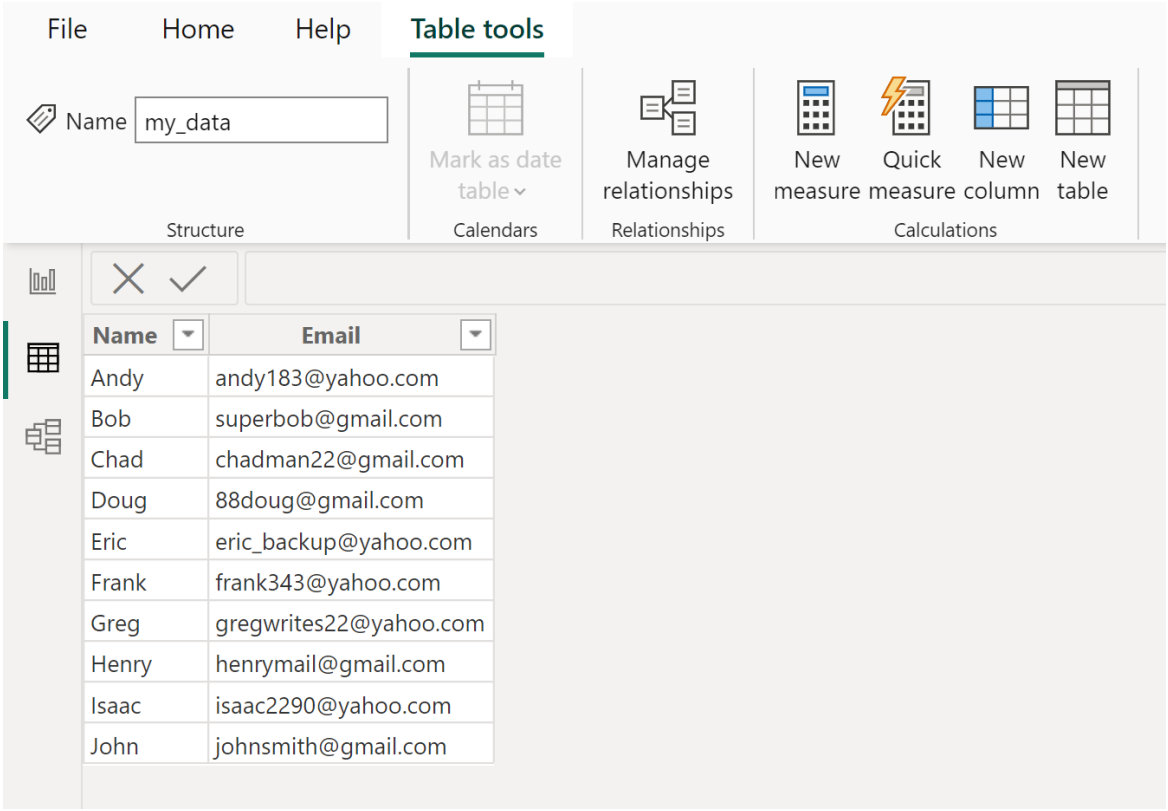
Formula 3: Extract Substring from End of String (Using RIGHT)

last_three = RIGHT('my_data', 3)

The RIGHT function serves as the counterpart to the LEFT function, enabling the extraction of characters starting from the end (rightmost side) of the string. Like LEFT, it requires only two arguments: the column reference and the count of characters to retrieve. This is commonly used when extracting file extensions (e.g., .pdf, .jpg) or specific suffix codes. The formula shown retrieves the final three characters found in the **Email** column data, counting backward from the last character.

Practical Application 1: Extracting Segments by Position

To fully grasp how these DAX functions operate, we will now apply them sequentially to a common sample dataset. The following table, **my_data**, provides the source data for all subsequent examples and demonstrates the structure upon which these calculations are based.



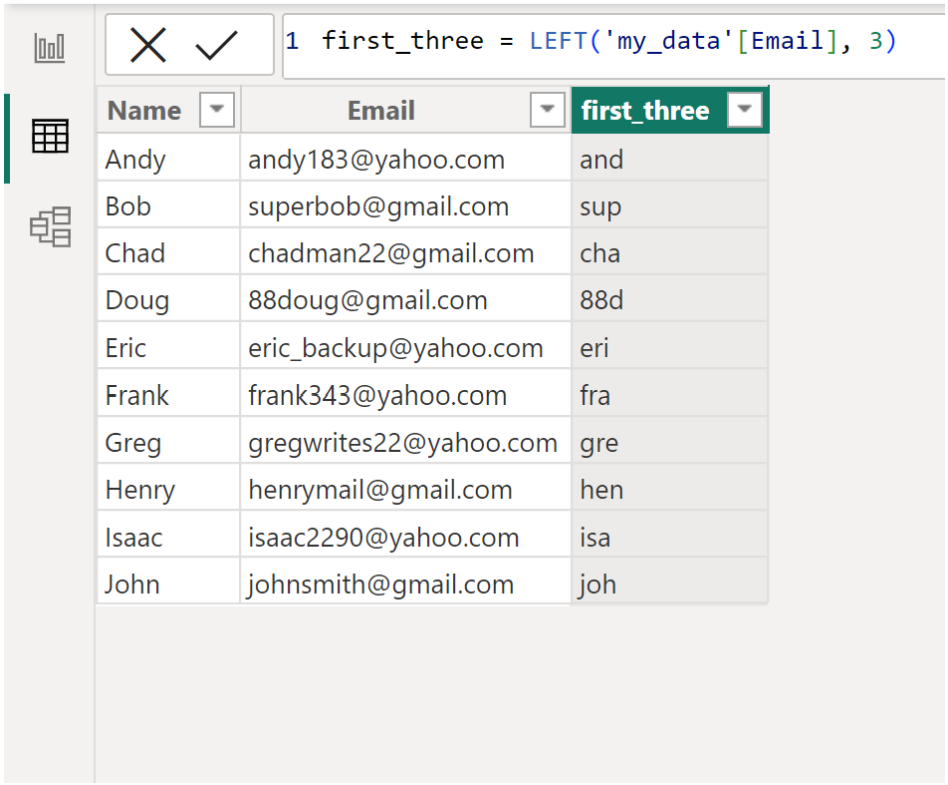
The process for creating these calculated fields in Power BI Desktop is uniform: navigate to the **Table tools** tab, select **New column**, and input the desired DAX expression into the formula bar. This action creates a new column where the formula is evaluated row by row, ensuring that each resulting value is contextually accurate for that specific record.

Example 1: Isolating Initial Characters (LEFT)

Our first goal is to isolate the initial three characters from every entry in the **Email** column. This is a simple but powerful technique for generating standardized, abbreviated identifiers.

first_three = LEFT('my_data', 3)

Executing this formula creates a new column named **first_three**. Notice how the calculation processes each row independently, returning only the leftmost three characters regardless of the total length of the original string. This consistency makes the LEFT function reliable for fixed-length prefix extraction.

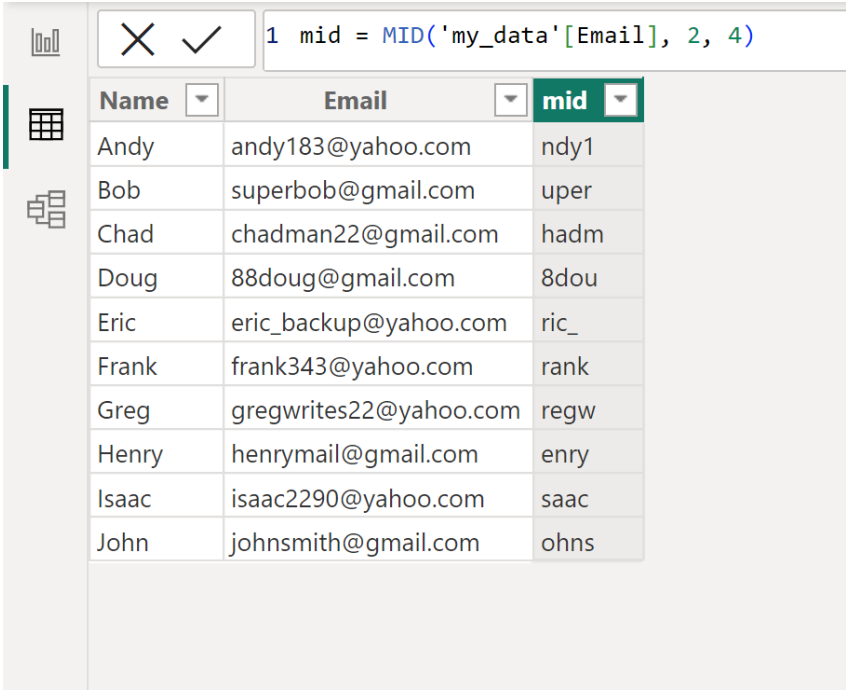


Example 2: Targeting an Internal Segment (MID)

To demonstrate the precision of the MID function, we aim to extract an internal 4-character sequence, specifically starting from the second position of the string. This is particularly useful when dealing with unique identifiers where specific digits or letters are known to reside in fixed middle positions.

mid = MID('my_data', 2, 4)

The resulting column, **mid**, confirms the function's behavior: it skips the very first character (position 1) and then returns the subsequent four characters. Analysts must be cautious when using MID on strings of varying lengths, as a specified starting position might fall outside the bounds of a shorter string, potentially leading to errors or unexpected results if not handled carefully.



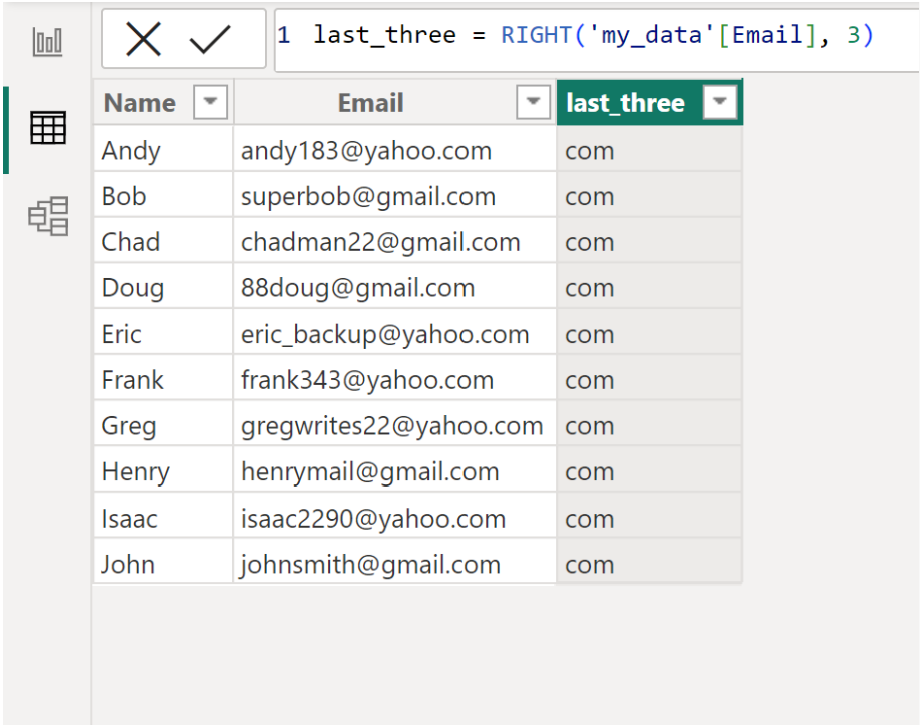
Name	Email	mid
Andy	andy183@yahoo.com	ndy1
Bob	superbob@gmail.com	uper
Chad	chadman22@gmail.com	hadm
Doug	88doug@gmail.com	8dou
Eric	eric_backup@yahoo.com	ric_
Frank	frank343@yahoo.com	rank
Greg	gregwrites22@yahoo.com	regw
Henry	henrymail@gmail.com	enry
Isaac	isaac2290@yahoo.com	saac
John	johnsmith@gmail.com	ohns

Example 3: Extracting Trailing Characters (RIGHT)

Finally, we use the RIGHT function to isolate the last three characters in the **Email** column. This is a straightforward operation that confirms our understanding of positional extraction from the right side.

last_three = RIGHT('my_data', 3)

The resulting column, **last_three**, shows the final characters of each email string. These three core positional functions (LEFT, MID, RIGHT) form the bedrock of DAX text handling, providing robust tools for fixed-length data parsing.



Advanced DAX Techniques: Dynamic Extraction Using Delimiters

While positional extraction is effective for fixed-width data, most real-world strings (such as names, addresses, or email addresses) have highly variable lengths. In these dynamic scenarios, relying on fixed counts is impossible. Instead, we must identify segments based on the location of a specific character, known as a delimiter, such as a space, comma, or the "@" symbol.

To achieve this dynamic extraction in Power BI, we must combine the positional functions (LEFT and RIGHT) with powerful analytical functions like SEARCH and LEN. The SEARCH function is critical here, as its primary purpose is to locate the exact starting position of a specified text string or character within another string. This position is then used as a dynamic length parameter for LEFT or RIGHT.

The primary challenge in dynamic extraction is calculating the correct length of the desired substring without including the delimiter itself. This is where the LEN function comes into play. LEN calculates the total number of characters in a string. By subtracting the position of the delimiter (found via SEARCH) from the total length (found via LEN), we can accurately determine how many characters follow the delimiter, allowing the RIGHT function to perform its retrieval correctly. These complex, nested functions are essential tools for professional data preparation.

Practical Application 2: Isolating Text Based on Delimiters

We now apply these combined techniques to our **Email** column, using the "@" symbol as the definitive delimiter to separate the username from the domain name. This demonstrates the power of DAX to handle real-world parsing challenges seamlessly.

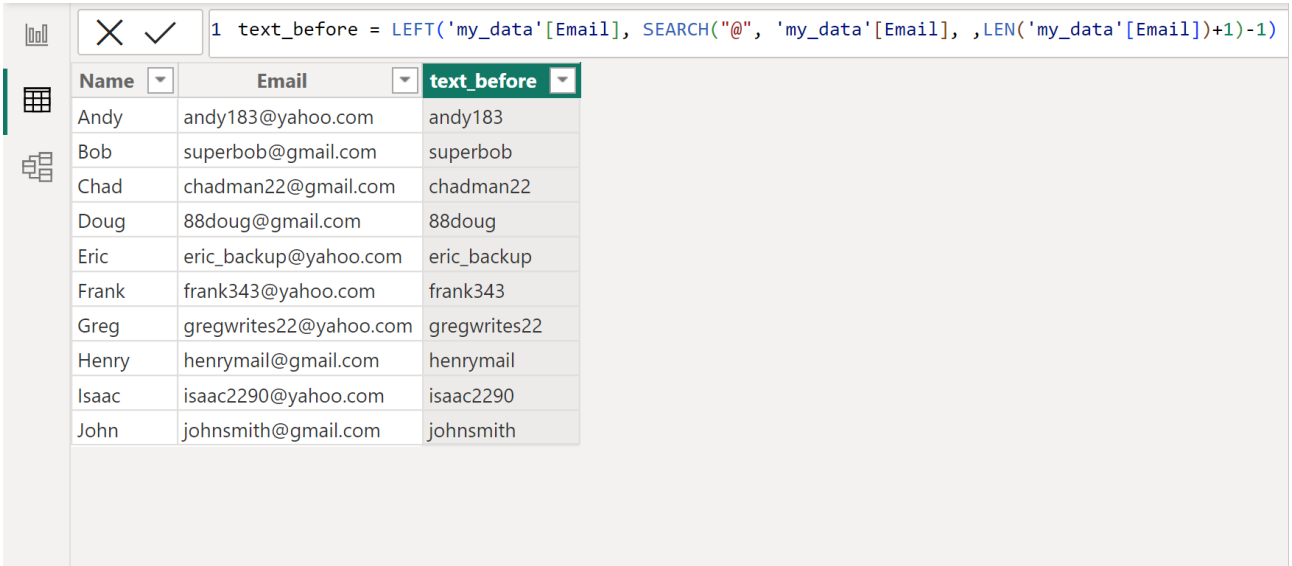
Formula 4: Extracting Text Before the Delimiter (Finding the Username)

To successfully isolate the username--the text segment preceding the "@" symbol--we must first locate the symbol's position and then feed that position into the LEFT function, adjusted by minus one to exclude the delimiter itself.

text_before = LEFT('my_data', SEARCH("@", 'my_data', ,LEN('my_data')+1)-1)

This powerful, combined formula performs three key steps: first, the inner SEARCH function locates the starting position of the "@" symbol. Second, we subtract 1 from this position, which gives us the exact count of characters in the username. Third, the outer LEFT function uses this calculated count as its second argument, extracting the precise username portion for every row, regardless of the username's length.

The calculated column, **text_before**, successfully isolates the username component, demonstrating a clean and robust dynamic extraction:



Name	Email	text_before
Andy	andy183@yahoo.com	andy183
Bob	superbob@gmail.com	superbob
Chad	chadman22@gmail.com	chadman22
Doug	88doug@gmail.com	88doug
Eric	eric_backup@yahoo.com	eric_backup
Frank	frank343@yahoo.com	frank343
Greg	gregwrites22@yahoo.com	gregwrites22
Henry	henrymail@gmail.com	henrymail
Isaac	isaac2290@yahoo.com	isaac2290
John	johnsmith@gmail.com	johnsmith

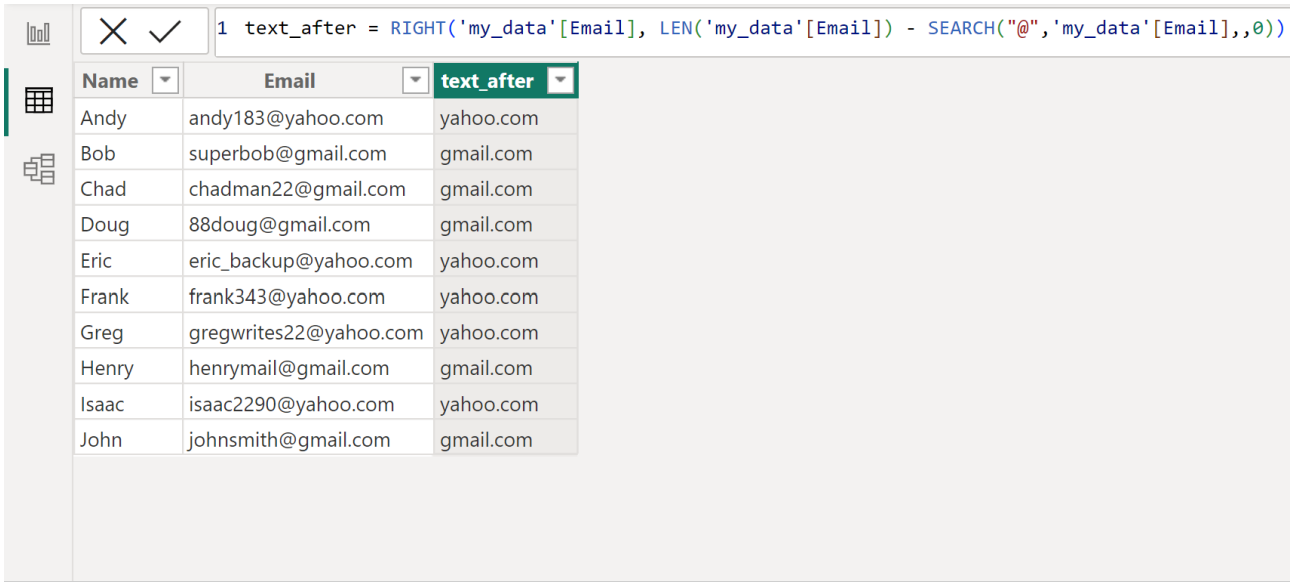
Formula 5: Extracting Text After the Delimiter (Finding the Domain)

Retrieving the domain name--the segment following the "@" symbol--requires a slightly different calculation structure using the RIGHT function. Because the RIGHT function counts from the end, we need to calculate the length of the string segment we wish to keep.

text_after = RIGHT('my_data', LEN('my_data') - SEARCH("@",'my_data',,0))

This formula is highly efficient: it first finds the total length of the string using the LEN function, then finds the position of the delimiter using SEARCH. Subtracting the position of the "@" symbol from the total length yields the exact number of characters that follow the delimiter. This resulting number is then supplied to the RIGHT function, completing the domain extraction with perfect accuracy across strings of varying lengths.

The final calculated column, **text_after**, successfully isolates the domain name for each entry, providing a structured, separate field for domain analysis:



Name	Email	text_after
Andy	andy183@yahoo.com	yahoo.com
Bob	superbob@gmail.com	gmail.com
Chad	chadman22@gmail.com	gmail.com
Doug	88doug@gmail.com	gmail.com
Eric	eric_backup@yahoo.com	yahoo.com
Frank	frank343@yahoo.com	yahoo.com
Greg	gregwrites22@yahoo.com	yahoo.com
Henry	henrymail@gmail.com	gmail.com
Isaac	isaac2290@yahoo.com	yahoo.com
John	johnsmith@gmail.com	gmail.com

Conclusion and Best Practices for DAX Text Functions

The ability to effectively parse and extract substrings using DAX is a fundamental requirement for advanced data preparation within Power BI. We have successfully demonstrated five essential formulas, ranging from simple fixed-position extractions using LEFT, MID, and RIGHT, to complex dynamic parsing achieved by combining these functions with SEARCH and LEN.

When implementing these techniques, always consider the variability of your source data. While positional functions are fast and simple, they should only be used on data sources known to be fixed-width. For all other scenarios, relying on the combined power of SEARCH/LEN with LEFT/RIGHT ensures that your data model remains resilient and accurate, even as underlying text data changes in length or structure. Furthermore, remember that calculated columns, while powerful for data preparation, consume memory (RAM) within the data model. Use these functions judiciously, and consider performing extensive data cleaning in Power Query (M language) first if

the extraction is purely for structural transformation rather than dynamic analysis.

Mastering text manipulation is key to unlocking the full potential of your raw data. For further exploration of data handling and transformation within the Power BI ecosystem, please consult the following tutorials and documentation:

Official Microsoft DAX Function Reference
Tutorials on Handling Errors and Blanks in DAX Calculations
Best Practices for Performance Tuning in Power BI Data Models